CMPT 117 (02)
Instructor: Kevin Grant

Midterm Examination
February 25, 2005

Name:

Student Number:

Rules:

1) You have 50 minutes to complete the exam. You have approximately one minute per mark, so allocate your time wisely
2) This is a closed book examination. You may not use any reference material (electronic or non-electronic) of any kind.
3) You may not communicate with others
4) Cheating is not tolerated, and will result in a forfeiture of the exam, and a 0% grade.

Good luck!!

Part I: Theory (10 Marks)

1. (4 Marks) Compare and contrast linked lists and dynamic arrays. That is, list two advantages of linked lists over dynamic arrays, and two advantages of dynamic arrays over linked lists.

linked lists: - don't require contiguous memory
- elements can be inserted without having to move everything before or after them.

dynamic arrays: - provide index notation to find something easily by position (don't have to iterate through n elements to find the n'th element)
- are self-contained (don't need to keep a bunch of head pointers, tail pointers, and node elements organized together.)

4

2. (2 Marks) Which uses more space, a 2-dimensional array from the call stack, or a 2-dimensional array from the heap? What is the difference in the amount of space used?

A 2D-array from the heap uses more. In addition to the memory used to hold the data elements, it must store an additional array of pointers to each row.

Difference in space used = nrows * sizeof(int*)

2

3. (1 Marks) List the major difference between a while loop and a do-while loop.

- a do-while loop is guaranteed to be executed at least once, as the check to continue is done at the end of the end of the first loop.

1

4. (3 Marks) What are the two ways to pass parameters to functions? What is the difference between these methods?

Parameters are passed by value or by reference.

↳ By value: a copy of the passed object is made & used locally within the function. The value of the object in the calling function is not changed.

↳ By reference: the address in memory of the passed object is sent to the function, so that any changes made to the object within the function affect the object in the calling function. (same)

3

Part II: Code Tracing (5 Marks)

Often in debugging, many programmers (who do not have access to Visual Studio) will leave little *cout* commands to help them find errors in code. In the following code segments, *cout* commands have been left in order to follow what the code is doing.

a) (3 marks) What would the output of this code be?

```
int f1(int x) {
   cout << "Calling f1(int)" << endl;
   return x * x;
}

double f1(double x) {
   cout << "Calling f1(double)" << endl;
   return x * x;
}

float f1(float x)  {
   cout << "Calling f1(float)" << endl;
   return x * x;
}

int main() {

   f1(2);
   f1(2.0f);
   f1(2.0);

   f1(f1(2) * f1(2.0));

}
```

Output:

Calling f1(int)
Calling f1(float)
Calling f1(double)
Calling f1(int)
Calling f1(double)
Calling f1(int) double

2.5

b) (2 marks) If there were no cout commands in our f1 functions, write a function template for f1 that would take the place of all three functions above.

2

```
template <class Item>
Item f1(Item x) {
    return x * x;
}
```

Part III: Linked Lists (10 marks)

Write a function that takes a pointer to the head of a linked list, and reverses the ordering of the nodes. The function should return void, and the head pointer of the list should be changed appropriately, to point at the new head of the list. The header file for the node class and linked list toolkit has been included at the back of your exam.

```
void reverse (node* & head_ptr) {

if (head_ptr == NULL)
    return;

size_t items = list_length (head_ptr);
node* new_head = list_locate (head_ptr, items-1) -> link();

for (size_t i = items-2; i >= 0; i--) {
 node* move_ptr = list_locate (i);
 value_type value = move_ptr.data;
 list_insert . . .
```

[out of time]

Part IV: (25 Marks) Object-Oriented Programming and Data Types

A single variable polynomial is of the form:

$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + \ldots\ a_nx^n$

We can represent this in a computer with a simple array. Each element a[i] in the array holds the coefficient a_i for each monomial in the polynomial. For instance, the polynomial:

$p(x) = 3 - 4x + 6x^3$

could be represented by the array:

{3, -4, 0, 6}

and

$q(x) = x + 2x^5$

as:

{0,1,0,0,0,2}

The *degree* of the polynomial is highest powered monomial. So for instance, the degree of p(x) is 3, and the degree of q(x) is 5.

On the following page is a *partially* completed polynomial class. The constructor for the polynomial class takes the degree of the polynomial. The constructor creates a new array (from the heap) to hold the monomial coefficients.

The overloaded operator [] returns the ith coefficient of the polynomial.

The function f takes a value *x_val*, and returns the value of the polynomial substituting *x_val* for x.

Here is how we would build the above polynomial p using our class.

```
Polynomial p(3);

p[0] = 3;
p[1] = -4;
p[2] = 0;
p[3] = 6;

cout << "p(1) = " << p.f(1); // outputs the value 5
cout << "p(2) = " << p.f(2); // outputs the value 43
```

The following questions are based on the completion of the polynomial class. You may find the following functions useful.

max(a, b) – returns the maximum between a and b
pow(a,b) – returns the value a^b

//polynomial.h

```
class Polynomial {
public:
      typedef double value_type;
      typedef size_t size_type;

      Polynomial(size_type deg = 0);
      ~Polynomial();


      size_type    getDegree() const;
      value_type   f(const value_type x_val) const;
      value_type & operator[](const size_type i);




private:
      value_type *coefficients;
      size_type degree;
};
```

//polynomial.cpp

```
Polynomial::Polynomial(size_type deg) {
   coefficients = new value_type[deg + 1];
   degree = deg;
}

Polynomial::getDegree(){
   return degree;
}

Polynomial::value_type &
Polynomial::operator[](const size_type i) {
  return coefficients[i];
}
```

a) (2 Marks) Add a destructor to the class. Write the appropriate prototype inside the class definition, and write the definition below.

```
Polynomial: ~Polynomial() {
    delete [] coefficients;
}
```

b) (6 Marks) Write the function definition for the f function.

```
value_type Polynomial::f(const value_type x_val) const {
    value_type result = 0
    for (int i = 0; i < degree; i++)
        result += coefficients[i] * pow(x_val, i);

    return result;
}
```

c) (5 Marks) The following function takes a polynomial as input, and returns a scaled version of that polynomial.

```
Polynomial scale(const Polynomial &p1, double scalar) {

    Polynomial result(p1.getDegree());

    for (size_t i = 0; i <= p1.getDegree(); i++)
       result[i] = p1[i] * scalar;

    return result;

}
```

(i) (2 Marks) Explain why this code won't compile.

(ii) (3 Marks) Add the appropriate code to the class so that the above function would compile. Write the function prototype in the class header, and the function definition in the space below.

d) (7 marks) Write an overloaded operator + for the Polynomial class. The operator should take two constant Polynomial parameters, and return a polynomial representing their sum. To add two polynomials, just pairwise add their coefficients. So for instance, if $p(x) = 3 - 4x + 6x^3$ and $q(x) = x + 2x^5$, then $p + q = 3 - 3x + 6x^3 + 2x^5$.

```
Polynomial operator + (Polynomial& p1, Polynomial& p2) {
    Polynomial result (max(p1.getDegree(), p2.getDegree());
    for( int i=0 ; i < min(p1.getDegree(), p2.getDegree()), i++) {
        result[i] = p1[i] + p2[i];
    }
    if (p1.degree < p2.degree) {
        for(int i = p1.degree(); i < p2.degree(); i++)
            result[i] = p2[i];
    }
    else {
        for (int i = p2.degree(); i < p1.degree(); i++)
        result[i] = p1[i];
    }
    return result;
}
```

e) (5 Marks) Rewrite the Polynomial class definition as a class template, where the coefficient types are parameterized by a template parameter. Note that YOU DO NOT HAVE TO REWRITE THE FUNCTION DEFINITIONS (just the class header).

```
template <class Item>
class Polynomial {
public:
   typedef Item value_type;
   typedef size_t size_type;
   Polynomial (size_type deg = 0, Item sample);
   ~Polynomial;
   size_type getDegree() const;
   Item   f(const Item x_val) const;
   Item &  operator [] (const size_type i);
private:
   Item * coefficients;
   size_type degree;
};
```

Unification occurs through the angle bracket notation

// required to unify. sample is any variable of desired value_type